Architecture Microservices

Projet - Desouches Arthur & Seassal Théo

1 juillet 2018

Liens du projet sous GitHub :
- https://github.com/Artdes95/Architecture_Microservices
- https://github.com/Artdes95/Operation-Change

Application : Bureau de Change en ligne

Table des matières

Introduction

Notre projet se décompose en deux sous-projets, représentant chacun un microservice : la gestion des taux de change d'un coté, la gestion des opérations de change de l'autre. Ces deux sous-projets sont hébergés sur des repository GitHub distinct pour des raisons que nous expliciterons dans la partie relative à Docker.

Ainsi, les instructions de compilation et de lancement qui vont suivre concernent aussi bien l'un que l'autre des codes sources GitHub fournis plus haut.

(« Architecture_Microservices » correspond à la gestion des taux de change, « Operation-Change », comme son nom l'indique, à la gestion des opérations de change.)

Compilation - Execution

La méthode la plus simple pour compiler et exécuter ce projet est de cloner directement le projet GitHub dans un éditeur (type Eclipse, ou IntelliJ) puis de faire un build. Un run de la classe principale démarrera ensuite le microservice associé.

Cependant, de nombreuses autres options sont aussi possibles.

- Télécharger les sources, les compiler (Javac) puis faire un run.
- Utiliser Maven : faire un maven -package, puis faire un run du Jar sous « target ».
- Faire un build docker et lancer l'image docker associée.

Documentation Technique

Plusieurs technologies/solutions ont été utilisé pour la réalisation de ce projet. Voici les principales.

A) Maven

L'ensemble du projet à été développé à l'aide de la solution « Maven » d'apache, principalement pour faciliter les aspects de compilation et de dépendance du code. Ainsi, à chaque lancement d'un « clean install/package », la répertoire « target » (architecture par défaut Maven) se charge de construire un Jar incluant les dépendances et pouvant être directement déployé et lancé.

B) Spring Boot

Le projet repose dans son intégralité sur la solution « Spring Boot » afin de construire un projet Spring le plus simplement possible, sans se soucier de toutes les configurations et réglages complexes et fastidieux. Ainsi, de nombreux frameworks ont été directement importés sans aucune action nécéssaire et parmi eux notamment : Spring Data JPA pour l'accès simplifié a la base de données et H2 pour la base de donnée elle même.

C) GitHub

Le gestionnaire de source GitHub, récemment racheté par Microsoft, à bien sur été utilisé afin que les deux sous-projets soit sauvegardés, et surtout accessibles et utilisables de façon simultané par les deux membres du groupe.

D) Docker

Nous avons décidé de gérer la conteneurisation à l'aide de Docker de deux manières.

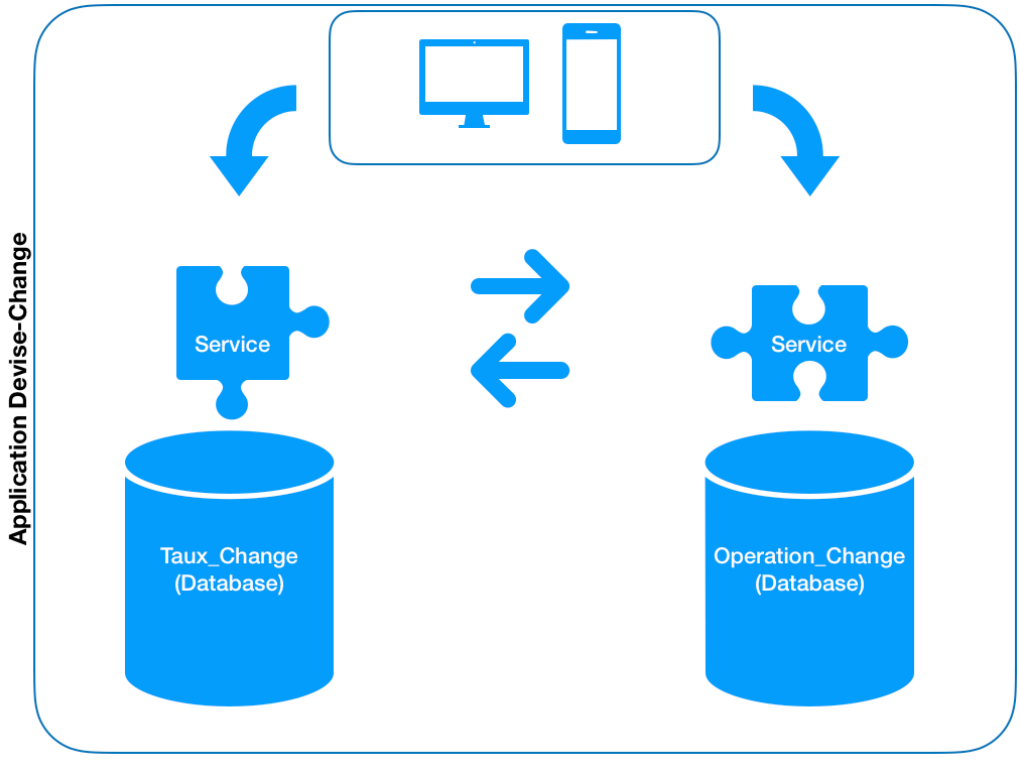
En local tout d'abord, à l'aide d'un plug-in docker : à chaque « build docker » la dernière version de l'image est construite et prête à être démarré à l'aide de docker (docker run).

Sur un « hub » docker en ligne :
- De façon manuelle à l'aide de la commande « docker push ».
- Mais aussi de façon automatique en associant au hub directement les repository Github de nos deux sous-projets. De cette manière, à chaque « push » du projet sur GitHub, le hub docker recompile automatiquement le projet et met à jour la dernière version de l'image docker.

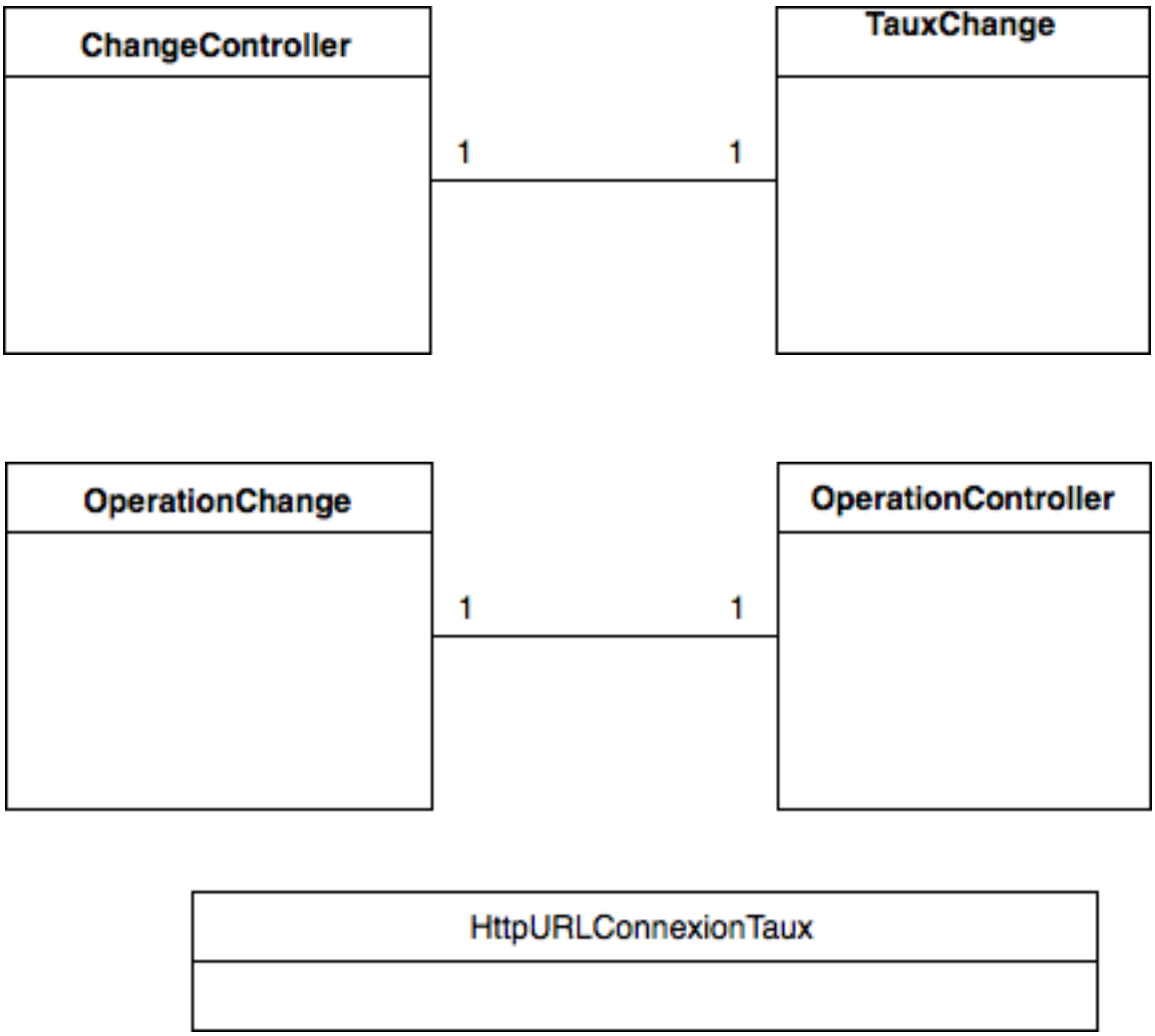
E) Postman

Enfin, nous avons utilisé l'application « Postman » (https://www.getpostman.com) afin de générer toutes sortes de requêtes HTTP, et d'automatiser celle-ci. En effet, les navigateurs classiques ne permettent pas la réalisation de requête de type « POST ».

Schéma d'architecture



Diagrammes de Classes



Bilan

Nous avons rencontré plusieurs difficultés lors de la réalisation de ce projet.

Tout d'abord, malgré la simplification apporté par Spring Boot, l'installation, la configuration et la prise en main des technologies nouvelles pour nous, en commençant par IntelliJ, fut très chronophage.
Ensuite, les deux membres du binôme ayant des connaissances basiques voire faibles en Java, la partie développement fut la difficulté majeures de notre travail.
Enfin, la partie Base de données n'étant pas non plus notre spécialité, nous avons du revoir nos connaissances en SQL et en gestion de base de données.

Cependant, nous avons beaucoup apprécié ce projet pour plusieurs raisons.

Tout d'abord, nous avons découvert de nombreuses solutions très pratiques, et facilitant grandement le travail à réalisé, et ce même pour un projet « from scratch ». Spring Boot bien sur, mais aussi de nombreux plug-in intelliJ, Maven et bien d'autres…

Ensuite, la conteneurisation Docker à été une véritable révélation. En effet, ayant travaillé dans un contexte de livraison d'application de production de façon « traditionnelle » c'est à dire par une équipe dédié, avec des processus lourd, la découverte de Docker nous a énormément plu. La possibilité de Build automatique avec la connexion GitHub, le lancement des images de façon simple, le plug-in IntelliJ pour la gestion de Docker etc…

Enfin, l'architecture en micro services et tout ce que cela implique nous a beaucoup intéressé. En effet, les différents environnements sur lesquelles nous avons travaillé ne suivent pas du tout ce principe, et cette solution nous semble une solution d'avenir qui mérite d'être appliqué par les systèmes que nous avons pu rencontrer au cours de nos années d'alternance.

Annexe : Services Disponibles

METHODE	PATH	DESCRIPTION
GET	/devise-change/taux	Retourne tout les taux de change
GET	/devise-change/taux/[id]	Retourne le taux de change identifié par id
GET	/devise-change/taux/[source]/[dest]	Retourne tout les taux de change identifié par source/dest
GET	/devise-change/taux/[source]/[dest]/[date]	Retourne le taux de change identifié par sa source sa destination et sa date
POST	/devise-change/taux/create	Création d'un taux de change (body)
POST	/devise-change/taux/modify	Modification d'un taux de change (body)
POST	/devise-change/taux/delete	Suppression d'un taux de change (body)
GET	/devise-change/operation	Retourne toutes les opération de change
GET	/devise-change/operation/[id]	Retourne l'opération de change identifié par son id
POST	/devise-change/operation/create	Création d'une opération (body)
POST	/devise-change/operation/modify	Modification d'une opération (body)
POST	/devise-change/operation/delete	Suppression d'une opération (body)
POST	/devise-change/operation/ createwithouttauw	Création d'une opération sans spécifié le taux de change qui est automatiquement récupéré de la table taux de change (par source, destination et date) (body)